Software Requirements Specification

for

Emergency Rescue Operation

1. Introduction

1.1 Purpose

The purpose of the Emergency Rescue Operation System is to provide a digital platform that enables efficient and effective management of emergency incidents, resources, and communication during various types of disasters or crises and Accident.

1.2 Scope

The system will allow users to report incidents, categorize them, allocate resources, monitor incident progress, and communicate in real-time. It will be accessible through web and, ensuring user-friendly interaction and responsiveness.

1.3 Overview of the SRS

This document outlines the functional and non-functional requirements of the Emergency Rescue Operation System. It describes the system's features, interactions, and constraints to guide the development process.

2. System Overview

2.1 Description of the System

The Emergency Rescue Operation System is a web-based application built on the Spring Boot framework for the backend and ReactJS for the frontend. It allows users to report incidents, allocate resources, track incident progress, and receive real-time notifications. The system supports different user roles, including administrators, emergency responders, and regular users.

2.2 High-Level Architecture

The system consists of a multi-tier architecture:

- Presentation Layer: ReactJS-based frontend for user interaction.
- Application Layer: Spring Boot-based backend for business logic and API management.
- Data Layer: MySQL database for storing incident, user, resource, and notification data.

2.3 User Roles and Responsibilities

- Administrator: Manages user accounts, resources, and system configuration.
- Emergency Responder: Reports incidents, allocates resources, and updates incident statuses.
- Regular User: Reports incidents, receives notifications, and views incident updates.

2.4 Interaction between Components/Modules

Components interact through RESTful APIs. The frontend communicates with the backend for user authentication, incident reporting, resource allocation, and notification retrieval.

2.5 Operating Environment

Core java, Spring boot
ReactJS
MySQL Db
Rest API ,Spring Security

3. Functional Requirements

3.1 User Management

- Users can register accounts with unique usernames and passwords.
- Users can log in using their credentials.
- Roles and permissions determine user access levels.
- Administrators can manage user accounts, including adding, editing, and deleting users.

3.2 Incident Reporting

- Users can report incidents, providing incident details, location, and severity.
- Incidents can be categorized during reporting.
- Users can attach media files (photos, videos) to incidents.

3.3 Incident Categorization

- Incidents are categorized based on predefined incident categories.
- Categories assist in incident prioritization and resource allocation.

3.4 Communication and Notification

- Users receive real-time notifications regarding incident updates.
- Notifications are sent via email, SMS, or push notifications.
- Emergency responders can communicate through instant messaging for collaboration.

3.5 Resource Management

- Resources include personnel, equipment, and supplies.
- Emergency responders can allocate resources to incidents.
- Resources have quantities and availability status.

Future Scope

3.6 Incident Tracking and Status

- Users can view incident statuses and progress.
- Emergency responders can update incident statuses and provide comments.

3.7 Reporting and Analytics

- The system generates reports on incident history, resource utilization, and response times.

4. Non-Functional Requirements

4.1 Performance

- The system should handle simultaneous incident reports and updates.
- Response times for critical actions should be within acceptable limits. ***4.2 Security***
- User passwords are stored securely using encryption techniques.
- Role-based access control ensures proper authorization.
- Data transmission is encrypted using HTTPS.

4.3 Reliability

- The system should have a high availability rate, with minimal downtime.
- Regular data backups and disaster recovery plans are in place.

4.4 Usability

- The user interface should be intuitive and user-friendly.
- Mobile responsiveness ensures usability on various devices.

5. System Constraints

- The system will be developed using Spring Boot for the backend and ReactJS for the frontend.
- The database will be MySQL for data storage.
- The application will be hosted on a cloud infrastructure.

6. Appendix A: Analysis Models